INTEGRIS

Date 02/17/97

Number of pages including cover sheet 7

TO:

Prof. Silvio Micali

Phone 739 3039

Fax 617 ~~547~~ ~~3118~~

FROM: Chini Krishnan
Integris Security, Inc.
333 W. El Camino Real, Suite 270
Sunnyvale, CA 94086

Phone 408 738 4935

Fax Phone 408 738 2159

CC: Dr. Martin Hellman

REMARKS: ☐ Urgent ☐ For your review ☐ Reply ASAP ☐ Please Comment

Hi Silvio,

Trust you are doing well. Dr. Hellman suggested I send you the enclosed paper: a short but lucid description of our technology along with some product datasheets.

After we talked, we realized faxing would be quicker. Please do let me know if you don't receive all pages.

Best Regards,

Chini

Quick Introduction to Certificate Revocation Trees (CRTs)

This document assumes a basic understanding of secure hash functions, certificates, certificate revocation lists, etc.

Introduction

When a party in a cryptographic protocol (such as a web browser or server) verifies a certificate, it must perform a series of tests to determine whether it is acceptable. One critical test is to determine whether the certificate has been revoked.

The traditional solution to the revocation problem involves certificate revocation lists. Each CA publishes signed lists of revoked certificates. The verifier downloads these lists, checks the signature on the list, makes sure the list is recent, the date of the list, and searches the list to make sure that the serial number of the certificate in question is not on the list. CRLs cause a variety of problems, most notably that the verifier must have an up-to-date list of all revoked certificates from the CA, a list which could potentially become very large.

CRTs make it possible for the verifier (any Internet or other application using certificates) to obtain a compact, time-critical proof that any certificate has not been revoked. A system based on CRTs consists of three major components: CRT issuance, confirmation issuance (of certificate status) and verification (of such confirmation).

CRT Issuance

To produce a CRT one must first obtain a set of all revoked certificates from the CA(s) in the tree. These would typically come from CRLs, but revocations could also come from other sources such as users revoking their own certificates. Actual revocation policies are determined by each CA.

Any revoked certificate can be uniquely identified by its CA's public key (or the hash of the public key) and the certificate serial number. For any CA, there may be zero or more revoked certificates.

Imagine a tree with three CAs with public key hashes $CA_1 < CA_2 < CA_3$, where CA_1 has revoked 4 certificates (156, 343, and 344), CA_2 has revoked no certificates, and CA_3 has 1 revoked certificate (987). The CRT issuer can now make the following statements about certificate serial number X from a CA whose public key hash is CA_X:

If: $-\infty < CA_X < CA_1$	Then: Unknown CA (revocation status unknown)
If: $CA_X = CA_1$ and $-\infty \le X < 156$	Then: X is revoked if and only if $X = -\infty$.
If: $CA_X = CA_1$ and $156 \le X < 343$	Then: X is revoked if and only if $X = 156$.
If: $CA_X = CA_1$ and $343 \le X < 344$	Then: X is revoked if and only if $X = 343$.
If: $CA_X = CA_1$ and $344 \le X < \infty$	Then: X is revoked if and only if $X = 344$.
If: $CA_1 < CA_X < CA_2$	Then: Unknown CA (revocation status unknown).
If: $CA_X = CA_2$ and $-\infty \le X < \infty$	Then: X is revoked if and only if $X = -\infty$.

If: $CA_2 < CA_X < CA_3$	Then: Unknown CA (revocation status unknown).
If: $CA_X = CA_3$ and $-\infty \le X < 987$	Then: X is revoked if and only if $X = -\infty$.
If: $CA_X = CA_3$ and $987 \le X < \infty$	Then: X is revoked if and only if $X = 987$.
If: $CA_3 < CA_X < \infty$	Then: Unknown CA (revocation status unknown).

Note that for any CA_X and X there is a single appropriate statement above which provides all known information about whether X is revoked. Furthermore, no statement can accidentally be applied to an inappropriate certificate.

The CRT issuer now builds a set of data structures which express the statements above. Note that only two basic types of data structures are required: those which specify a range of unknown CAs and those which specify a range of certificates in which the lower endpoint is revoked and all larger certificates in the range are valid.

The CA then builds a binary hash tree with hashes of the data structures as leaves. For the example above, there would be eleven leaf nodes (N_0..$N_{0,10}$), where $N_{0,i}$ is the secure hash of the structure expressing the i+1th statement above.

$$\begin{tikzcd}
N_{0,0} \arrow[r] & N_{1,0} \arrow[dr] & & & \\
N_{0,1} \arrow[ur] & & N_{2,0} \arrow[ddr] & & \\
N_{0,2} \arrow[r] & N_{1,1} \arrow[ur] & & & \\
N_{0,3} \arrow[ur] & & & N_{3,0} \arrow[dddr] & \\
N_{0,4} \arrow[r] & N_{1,2} \arrow[dr] & & & \\
N_{0,5} \arrow[ur] & & N_{2,1} \arrow[uur] & & \\
N_{0,6} \arrow[r] & N_{1,3} \arrow[ur] & & & N_{4,0} \\
N_{0,7} \arrow[ur] & & & & \\
N_{0,8} \arrow[r] & N_{1,4} \arrow[dr] & & & \\
N_{0,9} \arrow[ur] & & N_{2,2} \arrow[r] & N_{3,1} \arrow[uuur] & \\
N_{0,10} \arrow[r] & N_{1,5} \arrow[ur] & & &
\end{tikzcd}$$

Wherever two arrows converge on a single node (such as $N_{2,0}$ is produced from $N_{1,2}$ and $N_{1,3}$), the right-hand node is computed as the secure hash of the left-hand two nodes with the upper node hashed first. For example, $N_{2,0}$ would equal $H(N_{1,2} \mid N_{1,3})$, where "|" denotes concatenation. A nodes with a single arrow is equal to the node to its left (i.e., $N_{3,1} = N_{2,2}$). Note that the root node ($N_{4,0}$) is a function of all leaf nodes ($N_{0,0} \ldots N_{0,10}$). After producing the entire tree, the issuer digitally signs the root node along with supporting information (like the date and time of the tree issuance).

Confirmation Issuance and Verification

To determine the revocation status of a certificate, the verifier needs the appropriate statement and proof that the statement is correct. The role of a confirmation issuers is to provide a verifier these two components.

Expressing the statement is easy; that's just the appropriate leaf data structure. To assure the statement's validity, the verifier needs cryptographic proof that the leaf representative of the revocation status of the certificate is a the leaf in a properly signed tree. This assurance consists of the signed root node and a set of intermediate nodes which cryptographically bind the appropriate leaf node to the root node.

For example, for certificate 600 from CA_1, the appropriate statement is:

If: $CA_X = CA_1$ and $344 \le X < \infty$ Then: X is revoked if and only if $X = 344$.

The verifier can hash this statement structure to get $N_{0,5}$. The supporting nodes in this example are $N_{0,4}$, $N_{1,3}$, $N_{2,0}$, and $N_{3,1}$. The verifier can now use the secure hash function H to compute:

$$N_{1,2} = H(N_{0,4} \mid N_{0,5})$$
$$N_{2,1} = H(N_{1,2} \mid N_{1,3})$$
$$N_{3,0} = H(N_{2,0} \mid N_{2,1})$$
$$N_{4,0} = H(N_{3,0} \mid N_{3,1})$$

Finally, the verifier can check that $N_{4,0}$ was properly signed by the trusted tree issuer with the correct date, etc. If the original statement structure was tampered with or if any of the intermediate hashes were changed, the verifier will get a different value for $N_{4,0}$ than was $N_{4,0}$ signed by the tree issuer.

Advantages of a CRT based system

A CRT based approach has many advantages, notably:

- Verification does not require knowledge of entire CRLs.
- CRL caching and other optimizations are not required.

- Certificate holders can efficiently confirmations of their own certificates so that verifiers do not need to make any additional network connections.
- The system works with existing certificates, CRLs, etc.
- The size of the supporting hashes is approximately $20\log_2(N)$, where N is the number of revoked certificates. The system works efficiently even if N is in the billions or trillions.
- One tree with one signed root can provide proofs for all certificates in a chain, so only one public key operation can verify multiple certificates.
- The system supports CA revocation.
- All supporting information found in CRLs can be included including the revocation date/time, reason for revocation, etc.

DATA SHEET

Certium Server™

The Certium Server is an OEM application intended for integration with Certificate Servers that provides the ability to efficiently validate digital certificates to Internet applications. The server implements the tree-issuance and confirmation issuance portions of a certificate validity management system based on Certificate Revocation Trees.

The server includes C source code, developers' documentation and consulting assistance from Integris to complete the integration work.

BENEFITS

- Minimal network and processing overhead
- Scales to support billions of digital certificates
- Fully exportable
- Leading browser support planned

Highlights

- Comprehensive Intranet confirmation issuer for any Intranet Certificate Server. Primary features include:
  - a) Issuance of Certificate Revocation Tree based on both local certificate server or the globally operated Integris service.
  - b) Issuance of certificate confirmation to requesting applications.
  - c) Forms-based administration interface for easy addition to HTTP based Certificate Servers.
- An OEM product, source code available for tighter integration and customization.
- Implementation based on open standards: PKCS-7, RSA, SHA-1, X-509, ASN.1, HTTP.
- Single and multi-threaded implementations included.

DATA SHEET

Certium Application Toolkit™

The Certium application toolkit is a developer's toolkit that provides end user applications such as Web browsers and electronic mail clients with the ability to check the validity of digital certificates in real-time. The toolkit implements the verifier component of a certificate validity management system based on Certificate Revocation Trees. A typical verifier consists of functionality for the creation of certificate confirmation requests, the validation of certificate confirmation response, and the location of the nearest confirmation issuer.

The toolkit includes C source code, developers' documentation and consulting assistance from Integris to complete the integration work.

CERTIUM BENEFITS

- Certificate Validation transparent to end users
- Scales to support billions of digital certificates
- Fully exportable
- Leading browser support planned

Highlights

- Designed to work with both the Integris operated Certium Service and any Certificate Server or Certificate Management System that incorporates the Certium Server.
- Non-repudiable certificate status proof may be obtained by certificate holder, recipient or third-party.
- A transparent solution, end user typically unaware of validation.
- Thread-safe implementation, available on both Windows and UNIX.
- Cryptographic code bases supported include Tipem/BSAFE and SSLEay, CryptoAPI 2.0 planned for 4Q 1996 release.
- Implementation based on open standards: PKCS-7, RSA, SHA-1, X-509, ASN.1, HTTP.
- Toolkit license includes both development and redistribution rights.

01/23/1996 01:00 4087382159 INTEGRIS SECURITY PAGE 01

To: P. Muirhead
From: S. Micali
Pages: 13

MAIL ROOM MAR 13 1997 PAT. & TRADEMARK OFF.

14

INTEGRIS

Date 2/25/97

Number of pages including cover sheet 12

TO: Prof. Silvio Micali	FROM: Chini Krishnan, Integris Security, Inc., 333 W. El Camino Real, Suite 270, Sunnyvale, CA 94086
Phone	Phone 408 738 4925
Fax 617 739 3039	Fax Phone 408 738 2159

CC:

REMARKS: ☐ Urgent ☐ For your review ☐ Reply ASAP ☐ Please Comment

Hi Silvio,

Here is an ADN.1 spec that Paul requested I fax you.

I'm very delighted by the positive conversations you are having with Dr. Hellman & look forward to a constructive relationship!

Best Regards,

Chini

Feb 25, 97

Hi Don!

Please note that this fax was received Today while ~~the date indicated on top~~ is roughly a month earlier! ???

Best,

[illegible]

Please note that → the activity report of my Xerox 7020 shows Integris's fax as received today

RECEIVED

FEB 26 1997

F.H. & E. LLP PATENT DEPT.

FEB 25 '97 22:43 6177393039 PAGE.01

Certificate Validation Trees

DOCUMENT VERSION 1.05

INTEGRIS SECURITY, INC.
1230 OAKMEAD PARKWAY, SUITE 208
SUNNYVALE, CA 94086
TEL: (408) 738-2000
FAX: (408) 738 2159

1. Background

Under the X.509 directory infrastructure, certificate Revocation Lists (CRLs) are issued by Certificate Authorities (CAs) to revoke certificates. Because revoked certificates cannot be trusted, it is essential that systems using certificates include secure mechanisms to verify revocation status. Without such verification, a few compromised certificates could be used for wide-scale fraud. There will soon be millions of certificates accepted by protocols such as SSL and S/MIME and some fraction of these will need to be revoked.

Protocols should be optimized for the case where certificates are not revoked, since in practice revoked certificates will only be encountered in extraordinary circumstances (i.e., cases of attempted fraud). CRLs are extremely inefficient, since verification of a certificate's status requires the entire CRL. As a result, CRLs will become unacceptably large in large-scale systems involving millions of certificates. Furthermore, if certificate chaining and attribute certificates are used, users must obtain separate CRLs from each CA in the chain and every attribute certificate issuer. If CRLs are used for revocation, systems must be able to get CRLs from every CA. (This problem can be alleviated somewhat by designating CRL repositories, but these systems would consume an enormous amount of network bandwidth.) Complete new CRLs must be downloaded regularly since users cannot reliably determine whether a given certificate is valid without the complete CRL. Network traffic dedicated to CRL distribution could potentially exceed all other traffic combined, especially if CRLs are updated frequently.

While a variety of measures, such use of only very short-term certificates, can alleviate the situation somewhat, certificate validation trees eliminate the major problems associated with CRLs without sacrificing security. This document defines structures for certificate revocation trees.

2. System Overview

Certificate Validation Trees (CVT) trees provide an efficient way for certificate acceptors to determine whether certificates have been revoked and allow certificate holders to demonstrate that their own certificates have not been revoked. Trees are computed in advance by a "semi-trusted" party (such as Integris Security or a CA), which processes the data from one or more CRLs into tree form. The issuer is described as "semi-trusted," since the operation of the tree issuer is publicly auditable. Any independent party can verify that the tree issuer has not inappropriately revoked valid certificates or omitted revoked certificates from the tree. The tree's root node is then digitally signed.

To prove that one or more certificates have not been revoked, the tree leaves spanning the certificates in question are required, along with the supporting hashes binding the leaves to the tree's root and the digitally-signed root.

Certificate validation involves four kinds of entities: CAs, CVT issuers, confirmation issuers, and verifiers. The CAs are responsible for issuing certificates and issuing CRLs. A CVT issuer collects and verifies CRLs from one or more CAs then constructs a digitally-signed, dated certificate validation tree. The signed tree structure is then provided to confirmation issuers, which respond to users' requests for revocation information regarding specific certificates.

This document defines formats for the confirmation request, confirmation response, and CVT.

3. Data Formats

A **Confirmation Request** is passed from a verifier to a confirmation issuer and identifies one or more certificates whose revocation status is to be determined.

```
ConfirmationRequest        ::=    SEQUENCE {
        version                   Version DEFAULT v1,
        applicationID             INTEGER,
        hash                      AlgorithmIdentifier,
        requestList               SEQUENCE OF Request,
        requestExtensions         Extensions OPTIONAL }

Request                    ::=    SEQUENCE {
        issuerNameAndKeyHash      Hash,
        serialNumber              CertificateSerialNumber }

IssuerNameAndKey           ::=    SEQUENCE {
        issuer                    Name,
        issuerPublicKey           OCTET STRING } --Use what format!!!?--
```

The **issuerNameAndKeyHash** is computed by hashing an **IssuerNameAndKey** field constructed for the CA in question using a cryptographic hash function (i.e., SHA-1).

The confirmation issuer responds with a **ConfirmationResponse** structure:

```
ConfirmationResponse       ::=    SEQUENCE {
        resultStatus                      ResultStatus,
        certificateConfirmations          SEQUENCE OF
                                                  CertificateConfirmations,
        forwardingInfo              [1]   ForwardingInfo OPTIONAL
              -- Should make forwardingInfo an extension? !!! --
              -- NEED TO ADD [OPTIONAL] EXTENSIONS HERE--
}
```

```
ResultStatus              ::=  ENUMERATED {
        successful             ( 0 ),  --Response has valid confirmations--
        someCertsUnknown       ( 1 ),  --When is this returned?--
        malformedRequest       ( 2 ),  --Illegal confirmation request--
        requestorUnauthorized  ( 3 ),  --User not authorized to use issuer--
        internalError          ( 4 ),  --Internal error in issuer--
        serverHasNoTree        ( 5 ),  --Issuer tree missing or out of date--
        noGlobalTree           ( 6 )   --Request requires a global tree--
}

ForwardingInfo                 ::=  SEQUENCE {
        siteNameList           SEQUENCE OF OCTET STRING }

CertificateConfirmations       ::=  SEQUENCE {
        treeHeader             SIGNED { SEQUENCE {
                version            Version DEFAULT v1,
                minVersionToRead   Version DEFAULT v1,
                signature          AlgorithmIdentifier,
                issuer             GeneralNames,
                thisUpdate         GeneralizedTime,
                nextUpdate         GeneralizedTime,
                validUntil         GeneralizedTime,
                hash               AlgorithmIdentifier,
                totalLeafCount     LeafCount,
                rootHash           Hash,
                crtExtensions      Extensions OPTIONAL }},
        treeLeafAndPath        SEQUENCE SIZE (1..MAX) OF
                                       TreeLeafAndPath }

Version                   ::=  INTEGER { v1(0) }

TreeSerialNumber          ::=  INTEGER

LeafCount                 ::=  INTEGER (1..MAX)

Hash                      ::=  OCTET STRING
                               -- Length must equal hash size --

TreeLeafAndPath           ::=  SEQUENCE {
        treeLeaf               TreeLeaf,
        leafPath               SEQUENCE OF Hash }

-- TreeLeaf must be DER encoded. --
TreeLeaf                  ::=  SEQUENCE {
        leafPosition           LeafPosition,
        leafData                    LeafData }

LeafData                  ::=   CHOICE {
        unknownCA_Range         [0]     UnknownCARange,
        leafRange               [1]     LeafRange }

UnknownCARange            ::=   SEQUENCE {
        issuerPublicKeyHash1    Hash,
        issuerPublicKeyHash2    Hash }

LeafRange                 ::=   SEQUENCE {
        issuerPublicKeyHash     [0]     Hash OPTIONAL,
        thisCrlUpdate           [1]     GeneralizedTime OPTIONAL,
        nextCrlUpdate           [2]     GeneralizedTime OPTIONAL,
        validUntil              [3]     GeneralizedTime OPTIONAL,
        revocationDate          [4]     GeneralizedTime OPTIONAL,
        rangeMinStatus                  RevocationStatus,
        rangeStatus                     RevocationStatus,
        rangeMinimum            [5]     CertSerialNumber OPTIONAL,
        rangeMaximum            [6]     CertSerialNumber OPTIONAL,
        leafExtensions          [7]     Extensions OPTIONAL }

LeafPosition              ::=   INTEGER

RevocationStatus          ::=   ENUMERATED {
        revokedReasonUnspecified ( 0 ),
        keyCompromise           ( 1 ),
        cACompromise            ( 2 ),
        affiliationChanged      ( 3 ),
        superseded              ( 4 ),
        cessationOfOperation    ( 5 ),
        certificateHold         ( 6 ),
        expiredNormally         ( 7 ),
        noAcceptableCRL         ( 8 ),
        unsupportedRange        ( 32 ),
        unknownStatusOkay       ( 33 ),
        validCertificate        ( 64 ) }

CertSerialNumber ::= INTEGER
```

The treeHeader is digitally signed by the tree issuer and specifies the version of the current certificate validation tree, the oldest version with which the structure is backward-compatible, the tree serial number, the algorithm used to sign the header, the tree issuer's public key (hashed), the time of the tree's issuance, optionally the time of the next tree's issuance, optionally the time when the tree expires, the secure hash function used to construct the tree, the total number of leaves present in the tree, and finally the tree's root hash. The tree serial number values should be assigned sequentially by the tree issuer, beginning with zero. The tree issuer can either be a trusted third party entity (i.e., Integris Security) or the CA who originally issued the revoked certificates in the tree.

4. Verifying confirmations

[!!! This section has not been polished yet]

Every **TreeLeafAndPath** is a cryptographic assertion from the revocation tree issuer that no known revoked certificates exist within some range. Either the certificate in question is not known to be revoked, is revoked, is from a revoked CA, or is from a CA for which no CRL is present in the tree. (The third result is important, since applications might accept certificates from CAs whose CRLs are not in the tree, but no application should accept revoked certificates.)

To determine a certificate's status given a treeHeader and treeLeafandPath, do the following:

1. Check version and signature algorithm in treeHeader.
2. Verify that the issuer Name in the treeHeader corresponds to a trusted party or the CA who issued the certificate in question. If the name is unrecognized, the confirmation is invalid.
3. Check that the tree is acceptably recent by checking the header validUntil field. (See step 9 for checking of the leaf's time fields.)
4. Check that the Hash algorithm is acceptable (i.e., SHA).
5. Verify the signature on the treeHeader using the public key corresponding to the issuer's name.
6. Verify that leafPosition < totalLeafCount.
7. Using the leafPosition, verify that the leafPath binds the hash of the leaf to the rootHash in the signed treeHeader.
8. If the LeafData is of type UnknownCA_Range, verify that the hash of the CA's public key lies between issuerPublicKeyHash1 and issuerPublicKeyHash2. If so, the CA is not in the tree. If not, the leaf is inappropriate.
9. If the LeafData is of type LeafRange:
   - Verify that the issuerPublicKeyHash matches the hash of the CA's public key. (If issuerPublicKeyHash is omitted, it is identical to the tree issuer.) If the CA does not match, the leaf is inappropriate.
   - If present, verify that thisCrlUpdate, nextCrlUpdate, and validUntil are acceptable.
   - Verify that rangeMinimum, if present, is smaller than or equal to the serial number of the certificate in question. Also verify that rangeMaximum, if present, is larger than the serial number of the certificate in question.

- Check the RevocationStatus field. If $32 \leq$ RevocationStatus < 64, the certificate's status is unknown. The value 32 corresponds to an unknown range (i.e., absolutely no revocation information is available). The value 33 indicates that no current revocation information is present for this range, but this is normal so applications may decide to accept certificates in the range.
- If $64 \leq$ RevocationStatus < 96 or if certificateSerialNumber $\neq$ rangeMinimum, the certificate is valid. Otherwise the certificate is revoked for the reason specified in RevocationStatus.

The **issuerPublicKeyHash** values are computed as the hash of the DER-encoded public key of the CA. The time in **thisCrlUpdate** and optionally **nextCrlUpdate** in the LeafRange structure equal **thisUpdate** and **nextUpdate** from the CRL from the specified CA which is used in the tree.

The recipient of a **treeLeafAndPath** can use the **leafPath** to verify that each **treeLeaf** is cryptographically bound to the root:

```
Let maxDepth = $\lceil \log_2(\text{totalLeafCount}) \rceil$   -- $\lceil n \rceil$ rounds up if not an integer. --
Let y = leafPosition.
Let i = 0.
hash[0] = HASHED { treeLeaf }.
For x = 0 upto maxDepth-1
    Let y' = $(y \oplus 2^x) - ((y \oplus 2^x) \bmod 2^x)$.
    if y' < y then
        hash[x+1] = HASHED { hash[x] | leafPath[x-i] }.
    else if y' < totalLeafCount then
        hash[x+1] = HASHED { leafPath[x-i] | hash[x] }.
    else
        i = i + 1.
EndFor.
Assert (hash[maxDepth] = rootHash).
```

The operation of the revocation tree issuer can be verified easily, since anyone can verify that

1) all known revoked certificates and CAs should be contained in the tree, and
2) only known revoked certificates and CAs are contained in the tree.

A tree can include revocations for multiple CAs so that a single CertificateConfirmations structure can provide assurances for all certificates in a chain. The tree can even include revoked non-X.509 certificates (in which case the structure of TreeLeaf might change).

CertificateConfirmations messages are quite small; even if the tree includes many millions of revoked certificates. For example, each leafPath from a tree with a billion leaves using a hash size of 20 bytes would require about 600 bytes. A single asymmetric signature verification can provide assurances for all certificates in the chain.

Tree construction can be performed in linear time. Using the tree and signed root, confirmations can be constructed very efficiently using only insecure hardware.

5. ASN.1 Module

```
CertRevocationTreesDefinitions { joint-iso-ccitt (2) country (16) us (840)
organization (1) integris (--TBD!!!--) modules (1)
certRevocationTreesDefinitions (1) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS all definitions, in particular:
--      CertificateConfirmations,
--      -- Message syntax to communicate certificate confirmations --
--      ConfirmationRequest
--      -- Message syntax to request certificate confirmations --

IMPORTS
        AlgorithmIdentifier, SIGNED, Extensions
                FROM AuthenticationFramework {joint-iso-ccitt ds(5)
                module(1) authenticationFramework(7) 2}
                -- Note definition of Extensions requires application of
                -- Technical Corrigendum 1 --
        GeneralNames
                FROM CertificateExtensions {joint-iso-ccitt ds(5)
                module(1) certificateExtensions(26) 0} ;
-- Message syntax to request certificate confirmations --

ConfirmationRequest          ::=    SEQUENCE {
        version              Version DEFAULT v1,
        applicationID        INTEGER,
        hash                 AlgorithmIdentifier,
        requestList          SEQUENCE OF Request,
        requestExtensions    Extensions OPTIONAL }


Request                      ::=    SEQUENCE {
        issuerNameAndKeyHash        Hash,
        serialNumber                CertSerialNumber }
```

```
-- Message syntax to communicate certificate confirmations --
ConfirmationResponse       ::=    SEQUENCE {
        resultStatus                      ResultStatus,
        certificateConfirmations          SEQUENCE OF
                                                  CertificateConfirmations,
        forwardingInfo             [1]    ForwardingInfo OPTIONAL
}

ResultStatus               ::=    ENUMERATED {
        successful                 ( 0 ),
        someCertsUnknown           ( 1 ),
        malformedRequest           ( 2 ),
        requestorUnauthorized      ( 3 ),
        internalError              ( 4 ),
        serverHasNoTree            ( 5 ),
        noGlobalTree               ( 6 ) }


ForwardingInfo                    ::=    SEQUENCE {
        siteNameList              SEQUENCE OF OCTET STRING }

CertificateConfirmations          ::=    SEQUENCE {
        treeHeader                SIGNED { SEQUENCE {
                version           Version DEFAULT v1,
                minVersionToRead  Version DEFAULT v1,
                signature         AlgorithmIdentifier,
                issuer            GeneralNames,
                thisUpdate        GeneralizedTime,
                nextUpdate        GeneralizedTime,
                validUntil        GeneralizedTime,
                hash              AlgorithmIdentifier,
                totalLeafCount    LeafCount,
                rootHash          Hash,
                crtExtensions     Extensions OPTIONAL }},
        treeLeafAndPath           SEQUENCE SIZE (1..MAX) OF
                                          TreeLeafAndPath }

Version                    ::=    INTEGER { v1(0) }

TreeSerialNumber           ::=    INTEGER

LeafCount                  ::=    INTEGER (1..MAX)

Hash                       ::=    OCTET STRING
                                  -- Length must equal hash size --
```

```
TreeLeafAndPath          ::=   SEQUENCE {
        treeLeaf               TreeLeaf,
        leafPath               SEQUENCE OF Hash }

-- TreeLeaf must be DER encoded --
TreeLeaf                 ::=   SEQUENCE {
        leafPosition           LeafPosition,
        leafData               LeafData }

LeafData                 ::=   CHOICE {
        unknownCA_Range        [0]  UnknownCARange,
        leafRange              [1]  LeafRange }

UnknownCARange           ::=   SEQUENCE {
        issuerPublicKeyHash1   Hash,
        issuerPublicKeyHash2   Hash }

LeafRange                ::=   SEQUENCE {
        issuerPublicKeyHash    [0]  Hash OPTIONAL,
        thisCrlUpdate          [1]  GeneralizedTime OPTIONAL,
        nextCrlUpdate          [2]  GeneralizedTime OPTIONAL,
        validUntil             [3]  GeneralizedTime OPTIONAL,
        revocationDate         [4]  GeneralizedTime OPTIONAL,
        rangeMinStatus              RevocationStatus,
        rangeStatus                 RevocationStatus,
        rangeMinimum           [5]  CertSerialNumber OPTIONAL,
        rangeMaximum           [6]  CertSerialNumber OPTIONAL,
        leafExtensions         [7]  Extensions OPTIONAL }

LeafPosition             ::=   INTEGER

RevocationStatus         ::=   ENUMERATED {
        unspecified            ( 0 ),
        keyCompromise          ( 1 ),
        cACompromise           ( 2 ),
        affiliationChanged     ( 3 ),
        superseded             ( 4 ),
        cessationOfOperation   ( 5 ),
        certificateHold        ( 6 ),
        expiredNormally        ( 7 ),
        noAcceptableCRL        ( 8 ),
        unsupportedRange       ( 32 ),
        unknownStatusOkay      ( 33 ),
        validCertificate       ( 64 ) }
```

```
CertSerialNumber ::= INTEGER

END
```

[11] 4,309,569

[45] Jan. 5, 1982

[19]

References Cited

PATENT DOCUMENTS

Hellman et al. 375/2

Diffie

Ch, Test,